On The Computational Cost of FFT-Based Linear Convolutions

David H. Bailey

07 June 1996
Ref: Not published

Abstract

The "linear convolution" of two n-long sequences x and y is commonly performed by extending the input sequences with zeroes to length $2p$, where p is the smallest power of two greater than or equal to n, and then evaluating the circular convolution of these sequences using power-of-two FFTs. This approach clearly favors power-of-two input sequences sizes, with abrupt increases in computational cost when the size exceeds a power of two.

In this article, a recursive technique is presented for efficiently computing linear convolutions for any size n, even if one uses power-of-two FFTs as the underlying computational engine. The computational cost for this technique is never greater than the conventional approach and usually significantly less. Further, the computational cost as a function of n is highly continuous, so that linear convolutions of sizes somewhat larger than a power of two, for example, are only slightly more expensive than linear convolutions of power-of-two length data.

D. Bailey: NASA Ames Research Center, Mail Stop T27A-1, Moffett Field, CA 94035-1000.
E-mail: `dbailey@nas.nasa.gov`.

Introduction

Consider the problem of computing the "linear convolution" $\{z_k,\ 0 \le k < 2n\}$ of the n-long real sequences $\{x_k,\ 0 \le k < n\}$ and $\{y_k,\ 0 \le k < n\}$:

$$z_k \;=\; \sum_{0 \le i,j < n;\; i+j=k} x_i y_j \qquad\qquad 0 \le k < 2n$$

Convolutions of this form appear in many scientific applications, for example in solutions of large two- and three-dimensional partial differential equations [4, 7], and in fast schemes to perform multiple precision multiplication [2].

The linear convolution result can alternately be defined in terms of the "circular convolution", as follows. Extend the input sequences x and y with zeroes to length $2n$. Then the linear convolution of the original n-long sequences x and y is the same as the circular convolution of the extended $2n$-long sequences, namely

$$z_k \;=\; \sum_{j=0}^{2n-1} x_j y_{k-j} \qquad\qquad 0 \le k < 2n$$

where the subscript $k - j$ is interpreted as $k - j + 2n$ if negative.

Conventional Approaches

For the purposes of this discussion, we will assume that a linear convolution is evaluated, for larger n, by the usual scheme of extending the x and y sequences with zeroes as above, and then evaluating the resulting circular convolution using discrete Fourier transforms (DFTs) over the complex number field [3, pg. 85]:

$$z_k \;=\; F_k^{-1}[F_j(x) \cdot F_j(y)]$$

where $F_j(\cdot)$ denotes the $2n$-point forward DFT of x, and $F_j^{-1}(\cdot)$ denotes the inverse DFT. The notation $F_j(x) \cdot F_j(y)$ means that the two $2n$-long complex DFT result vectors are multiplied term by term, and the resulting $2n$-long complex vector is the input to the inverse DFT.

We will further assume that these DFTs are performed using fast Fourier transforms (FFTs) in an efficient manner. One commonly used efficient scheme is to compute the two forward FFTs using the real-to-complex variant of an FFT, and to compute the inverse FFT by using the complex-to-real variant [6, pg. 215-228]. The computational cost of either of these variants is approximately one half the cost of a complex-to-complex FFT of comparable size. Further, the result of a forward real-to-complex variant FFT is conjugate symmetric, so the term-by-term complex multiply operation indicated above need only be performed on $n + 1$ complex pairs instead of $2n$. Since, according to the conventional reckoning, a complex FFT of size 2^r requires $5r2^r$ floating-point arithmetic operations, we then have a total of $3 \cdot \frac{5}{2} \cdot (r+1)2^{r+1} + 6 \cdot (2^r+1) \approx (15r+21)2^r$ operations for the evaluation of a linear convolution on input sequences of length $n = 2^r$, using the above scheme.

This analysis assumes that the input size n is a power of two. Linear convolutions can be performed using the above scheme when n is not a power of two, provided that one extends the sequences x and y with zeroes to length $2p$, where p is the smallest power of two length greater than n, and performs FFTs of size $2p$. This is the usual approach.

Another approach is to use "mixed radix" FFTs, namely FFTs that transform data of sizes that are not purely powers of two, but include other integer factors such as three and five. This increases the number of admissible transform sizes, but most sizes remain inadmissible. More importantly, power-of-two FFTs are often the only FFTs available in the form of highly tuned vendor-supplied library routines. These are among the reasons that most FFT applications still employ power-of-two FFTs. Scientists computing linear convolutions thus generally assume that if one wants to compute the linear convolution of even $2^r + 1$ data points, one needs to use FFTs of size 2^{r+2}, which is double the FFT size (2^{r+1}) that suffices for linear convolutions of input sizes up to and including 2^r. In other words, the computational cost of this approach as a function of the input size n has abrupt discontinuities when n is one greater than a power of two, and is essentially flat between discontinuities.

A New Approach

However, there is a faster means of computing linear convolutions for input sizes sizes n that are not powers of two, even if one sticks to using power-of-two FFTs as the underlying computational engine. This can best be illustrated by means of an example. Consider the case $n = p + 2$, where $p = 2^r$ (so that x_p, x_{p+1}, y_p and y_{p+1} are potentially nonzero). First extend the sequences x and y with zeroes to length $2p = 2^{r+1}$ (instead of to length 2^{r+2} as in the conventional procedure for inputs of this size). Then apply forward and inverse FFTs to the extended sequences x and y to produce the circular convolution of these extended sequences, which is the following:

$$\begin{aligned}
z_0 &= x_0 y_0 + x_{p-1} y_{p+1} + x_p y_p + x_{p+1} y_{p-1} \\
z_1 &= x_0 y_1 + x_1 y_0 + x_p y_{p+1} + x_{p+1} y_p \\
z_2 &= x_0 y_2 + x_1 y_1 + x_2 y_0 + x_{p+1} y_{p+1} \\
z_3 &= x_0 y_3 + x_1 y_2 + x_2 y_1 + x_3 y_0 \\
\cdots &= \cdots \\
z_p &= x_0 y_p + x_1 y_{p-1} + \cdots + x_p y_0 \\
\cdots &= \cdots \\
z_{2p-1} &= x_{p-2} y_{p+1} + x_{p-1} y_p + x_p y_{p-1} + x_{p+1} y_{p-2}
\end{aligned}$$

This result differs from the desired $2n$-long linear convolution of x and y in two respects: (1) the initial three values (z_0, z_1 and z_2) are "corrupted" with some additional terms, and (2) the final four members of the sequence are missing. These four missing values are

$$\begin{aligned}
z_{2n-4} &= z_{2p} = x_{p-1} y_{p+1} + x_p y_p + x_{p+1} y_{p-1} \\
z_{2n-3} &= z_{2p+1} = x_p y_{p+1} + x_{p+1} y_p \\
z_{2n-2} &= z_{2p+2} = x_{p+1} y_{p+1} \\
z_{2n-1} &= z_{2p+3} = 0
\end{aligned}$$

Ignoring the last zero value, these three expressions are exactly the values that have "corrupted" the first three elements of the desired z sequence. Thus by separately computing these three expressions, one can correct the z sequence to the desired $2n$-point linear convolution result. Note that these three values can be obtained by computing a linear convolution on the sequences $\bar{x} = \{x_{p-1}, x_p, x_{p+1}\}$ and $\bar{y} = \{y_{p-1}, y_p, y_{p+1}\}$ and discarding the first two elements of the six-long result.

It is clear from this example that this technique can be extended to evaluate the linear convolution of sequences of size $n = p + m$ for any $m < p = 2^r$. First extend the n-long input sequences with zeroes to length $2p$, and obtain the $2p$-long circular convolution result by using FFTs. Then compute the "correction" sequence by performing a linear convolution on the $(2m-1)$-long sequences $\bar{x} = \{x_{p-m+1}, x_{p-m+2}, \cdots, x_{p+m-1}\}$ and $\bar{y} = \{y_{p-m+1}, y_{p-m+2}, \cdots, y_{p+m-1}\}$. From the resulting $(4m-2)$-long result sequence, discard the first $2m-2$ values as well as the final (zero) value, and then correct the $2p$-long sequence to the desired $2n$-long sequence as indicated above.

But clearly there is no point in doing this for m much larger than 2^{r-1}, because for this value of m the size of the correction convolution is roughly the same as the size of the 2^r-point convolution, and two linear convolutions of this size are nearly as expensive as a single linear convolution on inputs of size 2^{r+1}. In other words, the conventional approach of using FFTs of size 2^{r+2} is more efficient once m is larger than about 2^{r-1}.

The procedure that has been defined here can be applied recursively on the linear convolution of size $2m-1$, thus obtaining a recursive algorithm for linear convolutions that is never less efficient, and usually more efficient, than the conventional scheme of using FFTs of size 2^{r+2}. Of course one does not use the FFT-based scheme once $n < 64$ or so, since linear convolutions of such small sizes can be evaluated explicitly at lower cost than by applying FFTs. Note also that whenever a small linear convolution of size n is part of a larger linear convolution calculation being performed by the above scheme, only the second half of the result sequence needs to be computed — to be precise, only elements indexed $n-1$ through $2n-2$ need to be computed.

Computational Cost

Let $C(n)$ be the computational cost, in floating-point operations, of performing a linear convolution on inputs of size n using the above scheme, and let $D(n)$ be the cost of computing only the second half of the $2n$-long result sequence (see above for the precise statement of which values must be computed). Since the FFT procedure produces both halves of the result whether or not both are needed, $D(n) = C(n)$ whenever $n \geq 64$. When $n < 64$, however, we have $C(n) = n^2 + (n-1)^2$ and $D(n) = n^2$. We can now summarize this recursive algorithm as follows. To compute a linear convolution of size $n = 2^r + m$, where 2^r is the largest power of two not exceeding n:

If $n < 64$ then evaluate the linear convolution of x and y explicitly $[C(n) = n^2 + (n-1)^2]$, else if $n = 2^r$ then apply the conventional FFT-based scheme $[C(n) = (15r + 21)2^r]$, else apply the "correct" scheme described above $[C(n) = \min\{C(2^r) + D(2m-1) + 2m, C(2^{r+1})\}]$.

The function $C(n)$ is recursively defined by this algorithm statement. The graph of $C(n)$ is rather interesting — not only does it avoid the abrupt discontinuities of the conventional cost function, but it also has an intriguing fractal structure reminiscent of the Cantor function [5, pg. 48]. A plot of this function from $n = 65$ to $n = 65,536$ is shown in Figure 1.

This algorithm has recently been implemented in the Fortran-90 version of the author's multiprecision package [1]. This feature now removes a significant weakness of the "advanced" (extra-high precision) routines from this package, namely the restriction to precision levels that correspond to power-of-two vector lengths. Now the advanced routines can be efficiently used for a wide variety of precision levels. As expected, run times for various calculations follow the general form of the graph of $C(n)$ shown in Figure 1.

References

[1] David H. Bailey, "A Fortran-90 Based Multiprecision System", *ACM Transactions on Mathematical Software*, vol. 21, no. 4 (Dec. 1995), pg. 379–387.

[2] David H. Bailey, "Multiprecision Translation and Execution of Fortran Programs", *ACM Transactions on Mathematical Software*, vol. 19, no. 3 (Sept. 1993), pg. 288–319.

[3] William L. Briggs and Van Emden Henson, *The DFT: An Owner's Manual for the Discrete Fourier Transform*, SIAM, Philadelphia, 1995.

[4] Robert S. Rogallo, "Numerical Experiments in Homogeneous Turbulence", *NASA Ames Technical Memorandum*, TM81315, Appendix B, 1981.

[5] Halsey L. Royden, *Real Analysis*, Macmillan, New York, 1968.

[6] Charles Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, Philadelphia, 1992.

[7] Alan A. Wray and Robert S. Rogallo, "Simulation of Turbulence on the Intel Gamma and Delta", *NASA Ames Technical Memorandum*, TM91xxx, 1992.

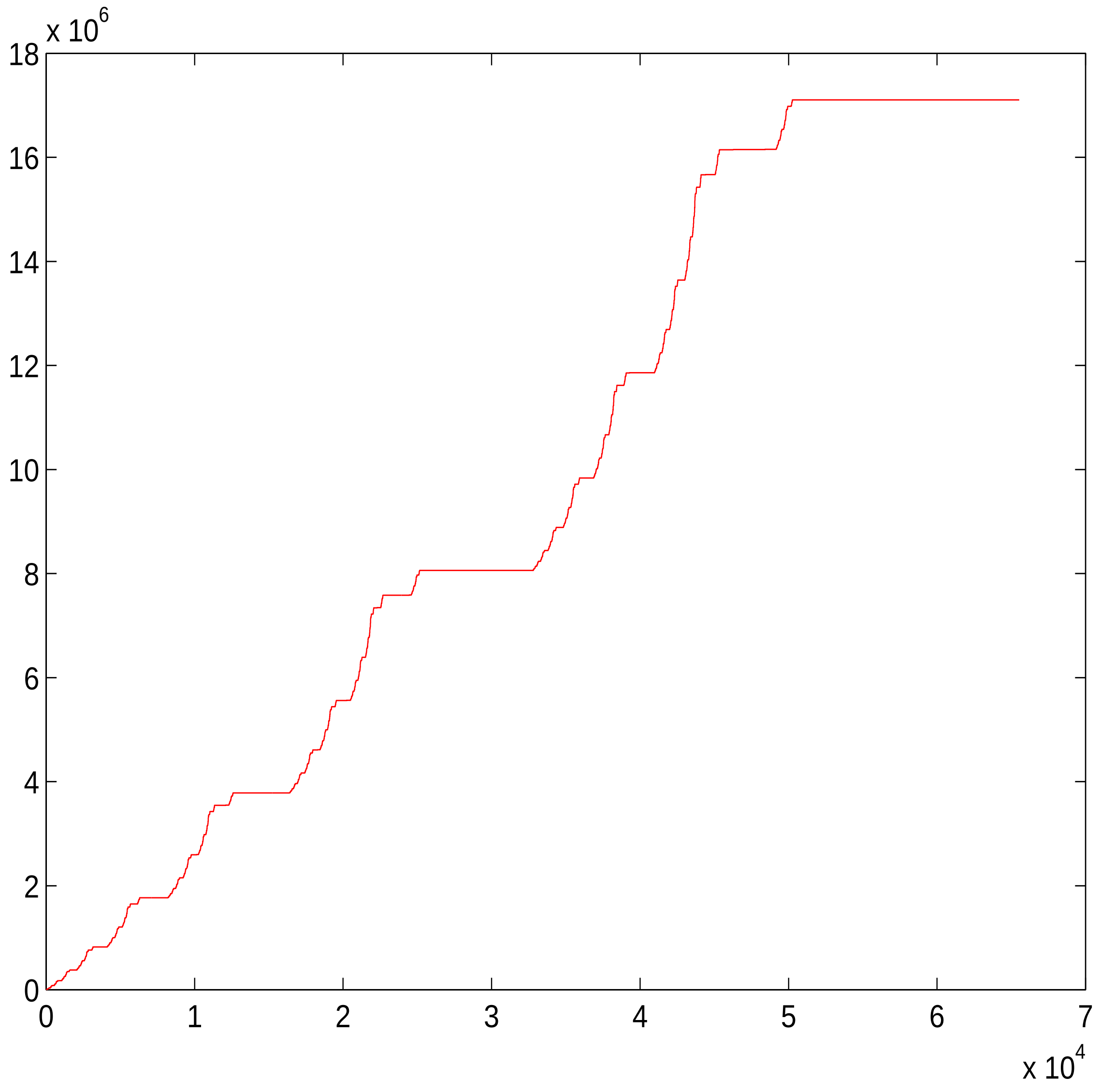

Figure 1: The Cost Function $C(n)$